CPET 499/ITC 250 Web Systems

Chapter 13
Managing State

Text Book:

*** Fundamentals of Web Development, 2015, by Randy Connolly and Ricardo Hoar, published by Pearson**

Purdue University Fort Wayne

Paul I-Hai Lin

Professor of Electrical and Computer Engineering Technology

http://www.etcs.ipfw.edu/~lin

Topics

- Why state is a problem in web application development
- What cookies are and how to use them
- What HTML5 web storage is and how to use it
- What session state is and what are its typical uses and limitation
- What server cache is and why it is important in real-world web sites.

The Problem of State in Web Applications

Figure 13.1 Desktop applications vs. web application

- All applications need to
  - Process user inputs
  - Output information, and
  - Read/write from databases or other storage media
- A web app consists of a series of disconnected HTTP request to a web server

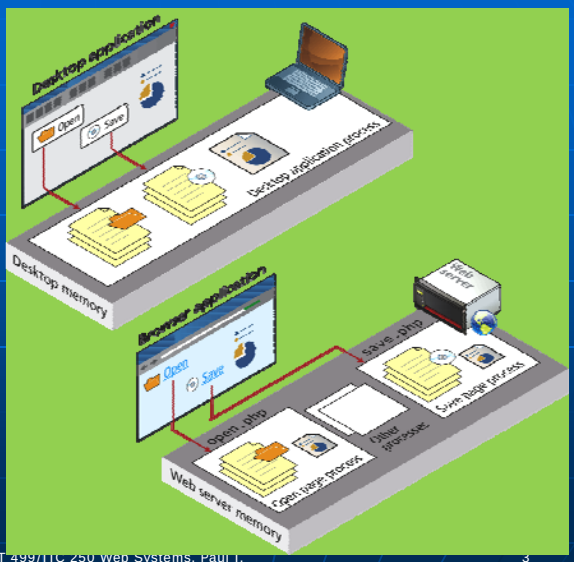


Figure 13.2 What the web server sees

- The web server sees only request
- The HTTP protocol does not without programming intervention, distinguish two requests

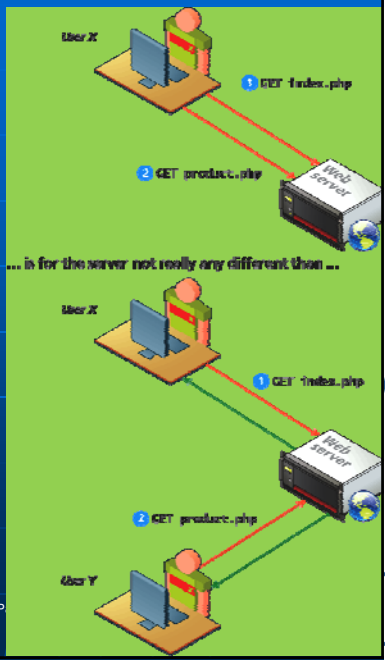
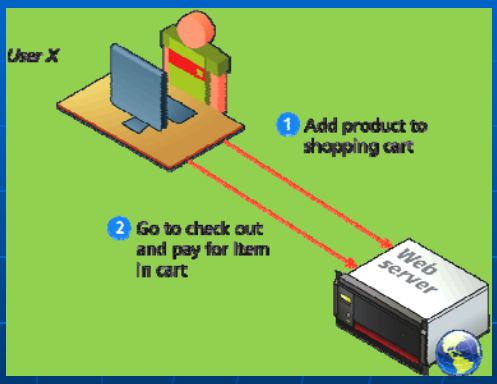


Figure 13.3 What the user wants the server to see

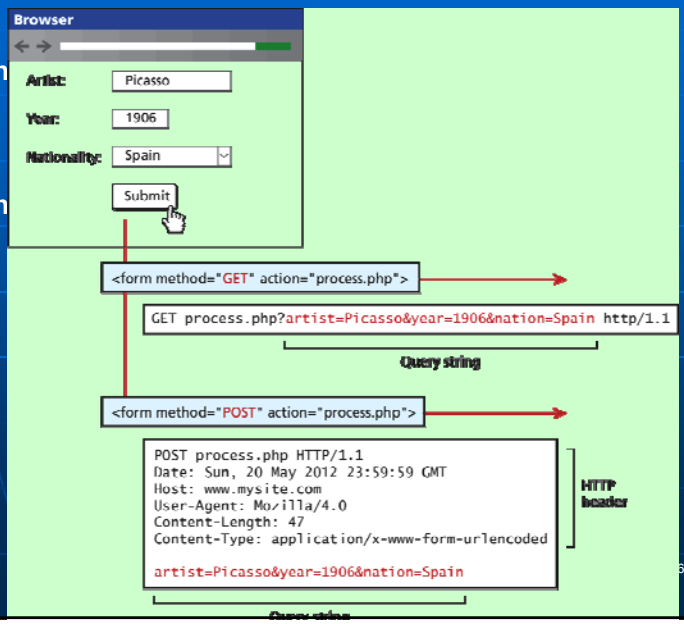
- User wants the web server to connect the request together: A web shopping cart example
- HTTP request-response interaction constrains information passing/using
- We can pass info using: Query strings, Cookies



Passing Information via Query Strings

Figure 13.4 Recap of Get vs. Post

- A query string within the URL (GET)
- A query string within HTTP header (POST)



Passing Information via the URL Path

- Drawbacks
  - The URL path and query string can be long and complicated
- For search engine application:
  - A prefer method
  - SEO (Search Engine Optimization)
  - Dynamic URLs (query string parameters) – an essential part of web development
  - URL Rewriting – a process of rewrite the dynamic URL into static one (and vice versa)
- Figure 13.5 URLs within a search engine result page

Figure 13.5 URLs with a search engine result page

Passing Information via the URL Path

- **Figure 13.5 URLs within a search engine result page**
  - **Top four commerce-related results for the search term "reproductions Raphael portrait la donna velata"**
  - **The top three: do not use query string parameters, use relevant info within the folder path or file name**
  - **File name extension is rewritten to make URL friendlier**
- **Rewrite URL**
  - **www.somedomain.com/DisplayArtist.php?artist=16**
  - **www.somedomain.com/artist/16.php**
- **More SEO friendly**
  - **www.somedomain.com/artist/Mary-Cassatt**
- **URL Rewriting in Apache and Linux**
  - **mod_rewrite module with .htaccess file**

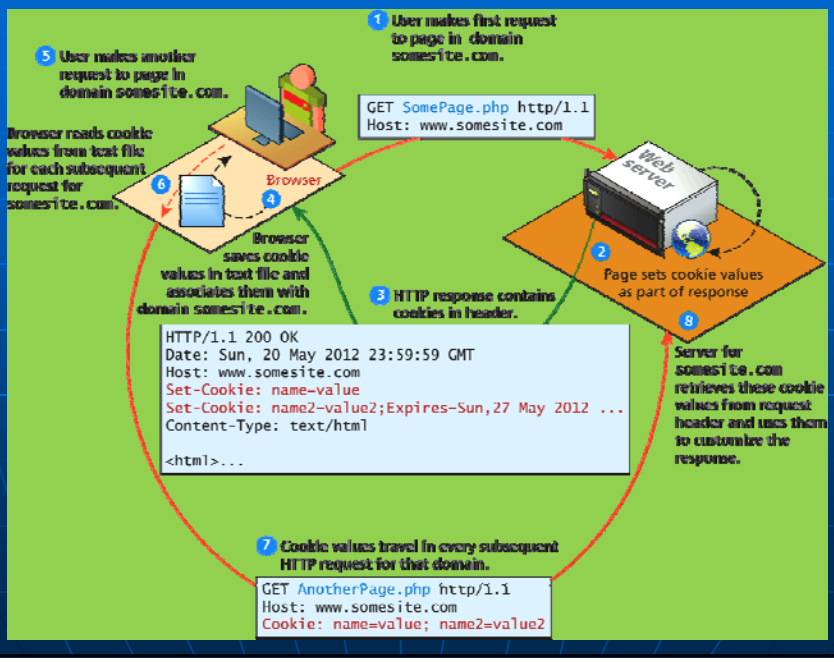
Cookies

- **HTTP Cookies:**
  - A client-side approach for persisting state information
  - Intended to be a long-term state mechanism used as a way of maintaining continuity over-time in a web application
  - They provide web servers with user-related information that can be stored on the user's computer and be managed by the user's browser
  - Also for keep tracking of whether a user has logged into a site
  - Storage space limitation – 4 k for a domain
  - IE 6 limited a domain to 20 cookies
  - Users can refuse to accept cookies

Cookies

- **Types of Cookies**
  - **Session Cookie** – no expiry state, will be deleted at the end of the user browsing session
  - **Persistent Cookies** – have expiry date specified
- Third-party tracking cookies – source of concern for privacy advocates
- Writing and Reading Cookies - PHP

Figure 13.6 Cookies at work



Cookies

- **Writing Cookies – PHP**

```
<?php
//listing 13.1 Writing a cookie
// add 1 day to the current time for expiry time
$expiryTime = time()+60*60*24;
// create a persistent cookie
$name = "Username";
$value = "Ricardo";
setcookie($name, $value, $expiryTime);
?>
```


Cookies

- **Reading Cookies – PHP**

```
<?php
//listing 13.2 Reading a cookie <-visit
Listing13.01.php to set the cookie.
if( !isset($_COOKIE['Username']) ) {
  //no valid cookie found
}
else {
  echo "The username retrieved from the cookie is:";
  echo $_COOKIE['Username'];
}
?>
```

Serialization

- **Serialization is the process of taking a complicated object and reducing it down to zeros and ones for either storage or transmission.**
- **PHP objects**
  - serialize() – reduce an object down to a binary string
  - unserialize() – reconstitute the binary string back into an object
- **Listing 13.3 the Serializable interface**

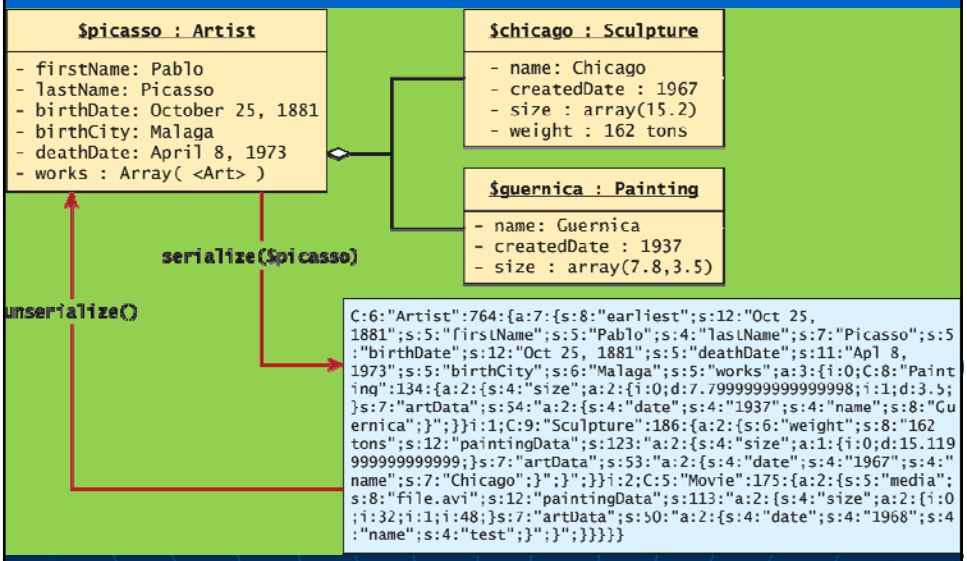
Serialization

- **Listing 13.3 the Serializable interface**

```
<?php
//listing 13.3 The Serializable interface
interface Serializable {
  /* Methods */
  public function serialize();
  public function unserialize($serialized);
}
?>
```

- **serialize($picasso);**
- **$picassoClone = unserialize($data);**

Figure 13.7 Serialization and deserialization



Listing 13.4 Art class modified to implement the Serializable interface

```
<?php
class Artist implements Serializable {
  //some parts borroed from earlier chapters.
  const EARLIEST_DATE = 'January 1, 1200';
  private static $artistCount = 0;
  private $firstName;
  private $lastName;
  private $birthDate;
  private $deathDate;
  private $birthCity;
  private $artworks;
```

Listing 13.4 Art class modified to implement the Serializable interface

```
// Implement the Serializable interface methods
  public function serialize() {
 // use the built-in PHP serialize function
    return serialize(
                    array("earliest" =>self::$earliestDate,
                          "first" => $this->firstName,
                          "last" => $this->lastName,
                          "bdate" => $this->birthDate,
                          "ddate" => $this->deathDate,
                          "bcity" => $this->birthCity,
                          "works" => $this->artworks
                          )
                    );
 }
```

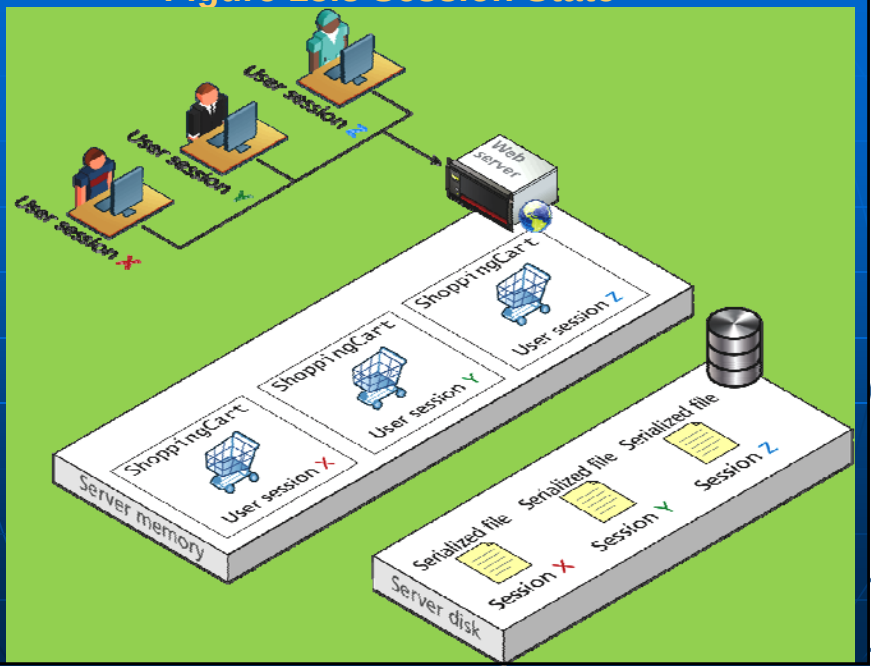

Listing 13.4 Art class modified to implement the Serializable interface

```
 public function unserialize($data) {
    // use the built-in PHP unserialize function
    $data = unserialize($data);
    self::$earliestDate = $data['earliest'];
    $this->firstName = $data['first'];
    $this->lastName = $data['last'];
    $this->birthDate = $data['bdate'];
    $this->deathDate = $data['ddate'];
    $this->birthCity = $data['bcity'];
    $this->artworks = $data['works'];
  }
  //...
}?>
```

Session State

- **Session state** – a server-based state mechanism that let web application store and retrieve objects for each unique session
- **Store serialized file on the server => deserialized and loaded into memory as needed for each request**
- **In PHP**
  - **Superglobal associative arrys**
  - **$_GET, $_POST, $_COOKIES**
  - **$_SESSION variable – needs additional steps to use**
- **See Figure 3.18**

Figure 13.8 Session State



Session State

- **Listing 13.5 Accessing session state**

```
<?php
//listing 13.5 Accessing session state
session_start();
if ( isset($_SESSION['user']) ) {
  // User is logged in
}
else {
  // No one is logged in (guest)
}
?>
```


Session State

- **Listing 13.6 Checking session existence**

```
<?php
//listing 13.6 Checking session existence
include_once("ShoppingCart.class.php"); //file not provided.
session_start();
// always check for existence of session object before
accessing it
if ( !isset($_SESSION["Cart"]) ) {
  //session variables can be strings, arrays, or objects, but
  // smaller is better
  $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

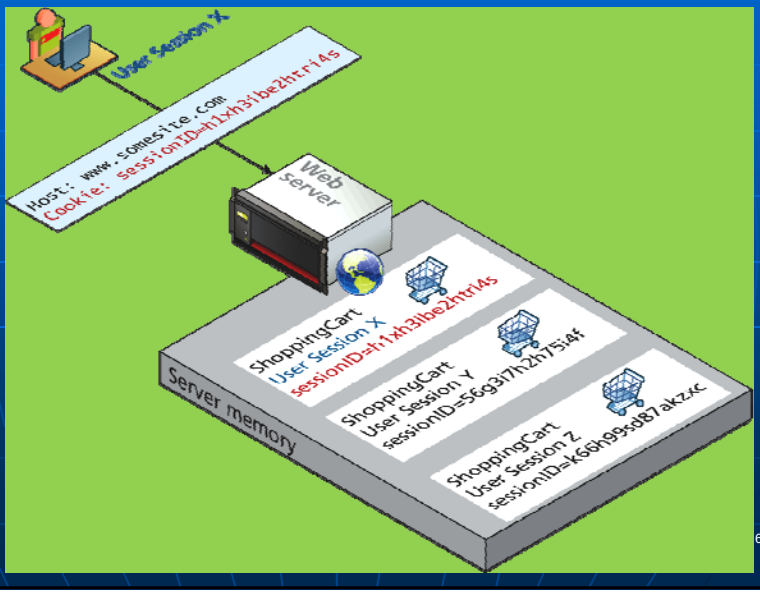
How Does Session State Work?

- HTTP is stateless
- Some type of user/session identification system is needed
- In PHP, see Figure 13-9
  - A session cookies
  - Server ⇔ a unique 32-byte string ⇔ User
- Listing 13.7 Configuration in php.ini to use a shared location for sessions

```
;listing 13.7 Configuration in php.ini to use a shared location for sessions
[Session]
; Handler used to store/retrieve data.
session.save_handler = memcache
session.save_path = "tcp://sessionServer:11211"
```

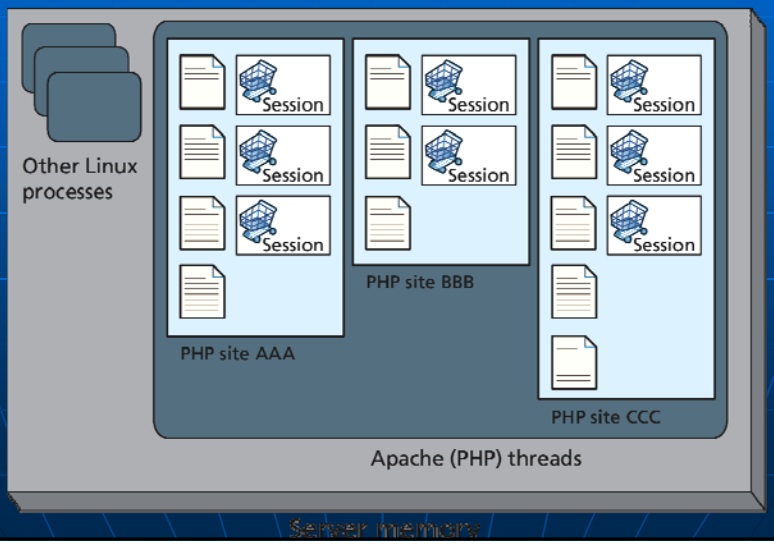

How Does Session State Work?

- Figure 13.9 Session ID



Session Storage and Configuration

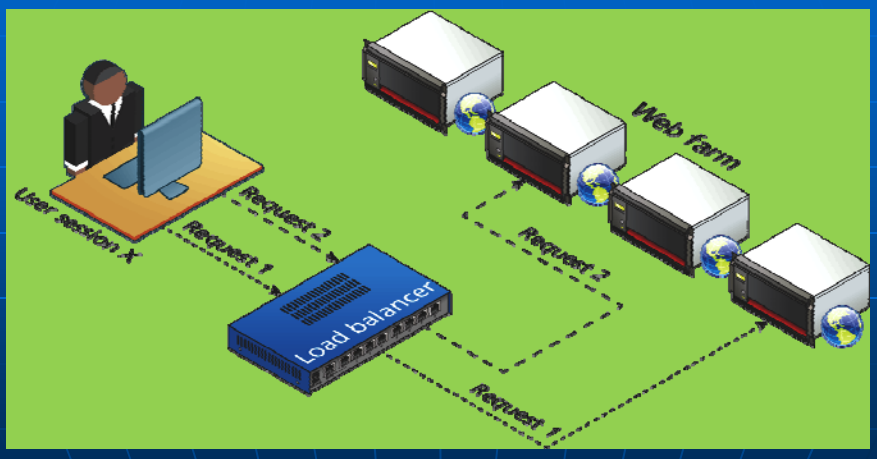
- Figure 13.10 Applications and server memory
  - **Store session info, pages being executed, and caching info**



27

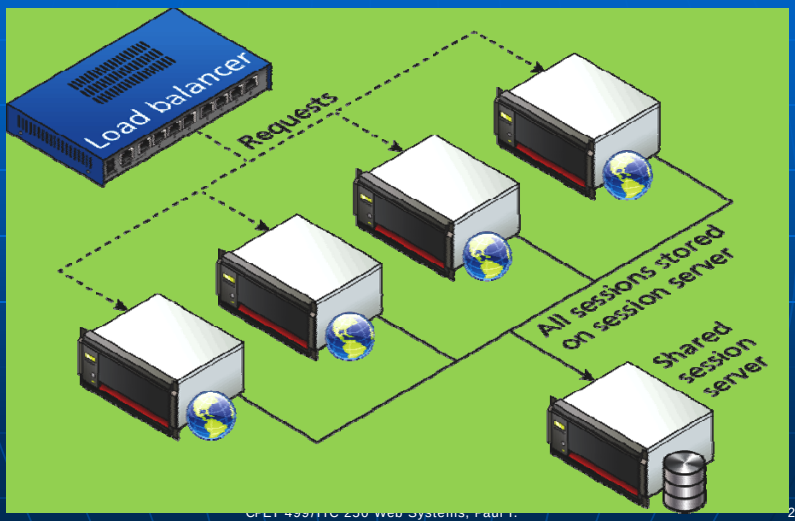
Session Storage and Configuration

- Figure 13.11 Web Farm



Session Storage and Configuration

- Figure 13.12 Shared session provider



HTML5 Web Storage

- Web storage – a new JavaScript-only API introduced in HTML5; managed by the browser
- It is meant to be a replacement (supplement) to cookies
- W3C recommends a limit of 5MB, but browsers are allowed to store more per domain.
- Should not be used for mission-critical application functions
- Using asynchronous communications via JavaScript to push the info to the server
- Two types of global web storage objects (key-value collections):
  - localStorage
  - sessionStorage

Listing 13.8 Writing web storage

```
<form ... >
<h1>Web Storage Writer</h1>
<script language="javascript" type="text/javascript">
if (typeof (localStorage) === "undefined" || typeof (sessionStorage)
=== "undefined") {
alert("Web Storage is not supported on this browser...");
}
else {
sessionStorage.setItem("TodaysDate", new Date());
sessionStorage.FavoriteArtist = "Matisse";
localStorage.UserName = "Ricardo";
document.write("web storage modified");
}
</script>
<p><a href="Listing13.09.html">Go to web storage reader</a></p>
</form>
```


Listing 13.9 Reading web storage

```
<form id="form1" runat="server">
<h1>Web Storage Reader</h1>
<script language="javascript" type="text/javascript">
if (typeof (localStorage) === "undefined" ||
typeof (sessionStorage) === "undefined") {
alert("Web Storage is not supported on this browser...");
}
else {
var today = sessionStorage.getItem("TodaysDate");
var artist = sessionStorage.FavoriteArtist;
var user = localStorage.UserName;
document.write("date saved=" + today);
document.write("<br/>favorite artist=" + artist);
document.write("<br/>user name = " + user);
}
</script> </form>
```

Why Would We Use Web Storage

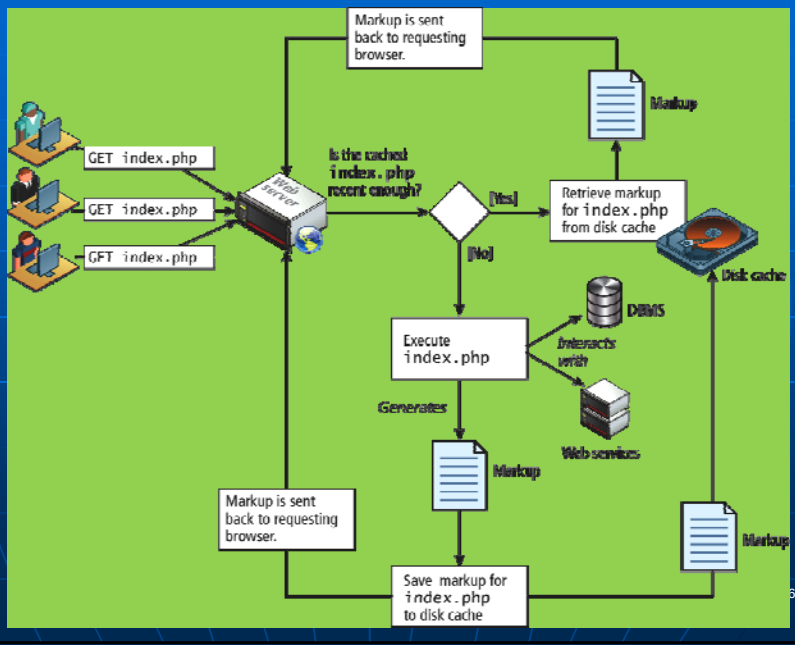
- Cookies Disadvantages
  - **Limit in size (4 k)**
  - **Being send in every single request-response to/from a given domain**
  - **Potentially disabled by the user**
  - **Vulnerable to XSS (Cross-Site Scripting) attack**
- Web Storage with JavaScript API
  - **Local cache for relatively static items available to JavaScript**
  - **One practical use: store XML or JASON from a web service to reduce server load for subsequent requests by the session**

Figure 13.13 Using web storage

Caching

- Using local storage
- A vital way to improve the performance of web applications
- HTTP protocol headers related to caching
  - Expires
  - Cache-Control
  - Last-Modified
- Two strategies to caching web applications
  - Page output caching
  - Application data caching

Fig 13.14 Page output caching



Listing 13.10 Using memcache for Application data caching

```
<?php

//listing 13.10 Using memcache
// create connection to memory cache
$memcache = new Memcache;
$memcache->connect('localhost', 11211) or die ("Could not connect to memcache server");
$cacheKey = 'topCountries';
/* If cached data exists retrieve it, otherwise generate and cache
   it for next time */
```


Listing 13.10 Using memcache for Application data caching

```
$countries = $memcache->get($cacheKey);
if ( ! isset($countries) ) {
  // since every page displays list of top countries as links
  // we will cache the collection
  // first get collection from database
  $cgate = new CountryTableGateway($dbAdapter);
  $countries = $cgate->getMostPopular();
  // now store data in the cache (data will expire in 240 seconds)
  $memcache->set($cacheKey, $countries, false, 240)
    or die ("Failed to save cache data at the server");
}
// now use the country collection
displayCountryList($countries);
?>
```

Summary and Conclusion

Q/A ?